יניב ביאליק מבחן ציונט

# Question 1

## GitHub Pull Request – A Pull Request Form GitHub asks a owner of the repository to give you permission to you pull all the files that pushethed into a repository / branch so you could change them.

## Inheritance – it’s a concept of object-oriented programming that let you make classes and a subclasese that Inherit variables and methods of their “father class” and can override them

## Dependency injection – it’s a design pattern that help us to detach classes that depend on each other. The dependency injection makes your classed loosely coupled by injecting the tdependency form out side that’s makes the classes less dependent and easier to unit test and make changes

**Example: we have a class that make a connection and a class of database X fi we will make the implementation of the connection depended on the database X it will be harder to change the database later if it will be needed so we want to make an interface that will inject the database X from outside into the connection**

## Refactoring - allows you to change the code without changing what the code does

## Exception Handling – to handle exceptions we should throw Exception in the method if it doing something that we don’t want it to do. In the main program we should try and catch the Exception and resolve the it

**f) Log Level – the Ilogger have built-in different importance or severity of log messages**

# Question 2

public static void MultiplicationBoard(int size)

{

string str = "";

for (int i = 1; i < size+1; i++)

{

for (int j = 1; j < size+1; j++)

{

str += " ";

String multi = (i \* j).ToString();

str += multi;

if (multi.Length == 1)

{

str += " ";

}

}

str += "\n";

}

Console.WriteLine(str);

}

**a)**

**b)**

public static void MultiplicationBoard(int row , int col)

{

string str = "";

for (int i = 1; i < row + 1; i++)

{

for (int j = 1; j < col + 1; j++)

{

str += " ";

String multi = (i \* j).ToString();

str += multi;

if (multi.Length == 1)

{

str += " ";

}

}

str += "\n";

}

Console.WriteLine(str);

}

# Question 3

public static void primeFactors(int n)

{

while (n % 2 == 0)

{

Console.Write(2 + " ");

n /= 2;

}

for (int i = 3; i <= Math.Sqrt(n); i += 2)

{

while (n % i == 0)

{

Console.Write(i + " ");

n /= i;

}

}

if (n > 2)

{

Console.Write(n);

}

Console.WriteLine();

}

internal class Program

{

static void Main(string[] args)

{

//Question2.MultiplicationBoard(10);

//Question2.MultiplicationBoard(10,5);

Question3.primeFactors(30); // = 2 3 5

Question3.primeFactors(100); // = 2 2 5 5

Question3.primeFactors(1024); // = 2 2 2 2 2 2 2 2 2 2

Question3.primeFactors(726); // = 2 3 11 11

Console.ReadLine();

}

}

public string primeFactorss(int n)

{

string str = "";

while (n % 2 == 0)

{

str += "2 ";

n /= 2;

}

for (int i = 3; i <= Math.Sqrt(n); i += 2)

{

while (n % i == 0)

{

str += i.ToString()+" ";

n /= i;

}

}

if (n > 2)

{

str += n.ToString();

}

return str;

}

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Exam;

namespace TestQuestion3

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void PrimeFactorsOf100()

{

string value = "100";

string actual = Question3.primeFactors(value);

string expected = "2 2 5 5";

Assert.AreEqual(expected, actual);

}

}

}

# Question 4

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Exam;

namespace TestQuestion4;

[TestClass]

public class UnitTest1

{

[TestMethod]

public void PromotedSimple()

{

string value = "555";

string actual = Question4.GetNextNumber(value);

string expected = "556";

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void Promotedlast9()

{

string value = "259";

string actual = Question4.GetNextNumber(value);

string expected = "260";

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void Promotedlast10are9()

{

string value = "699";

string actual = Question4.GetNextNumber(value);

string expected = "700";

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void IsDigitPromotedInTheMiddle()

{

string value = "99999";

string actual = Question4.GetNextNumber(value);

string expected = "100000";

Assert.AreEqual(expected, actual);

}

}

public static string GetNextNumber(string input, int End = 0)

{

if (End >= input.Length)

{

return "".PadLeft(input.Length, '0');

}

int digit = int.Parse(input[input.Length - 1 - End].ToString());

if (digit == 9)

{

return GetNextNumber(input, ++End);

}

else

{

return input.Substring(0, input.Length - 1 - End) + (++digit).ToString().PadRight(End + 1, '0');

}

}